Communication Security

Paul Mooney
Chief Software Architect, Microsoft MVP

@daishisystems | www.insidethecpu.com

Overview

Coming Up

- Communication security is vital for web application developers
- Encryption ensures confidentiality and integrity of data exchanges
- Encryption protects against Man-in-the-Middle (MITM) attacks
- Focus on securing HTTP/HTTPS and Websockets communication channels
- Understand security considerations and best practices for HTTP/HTTPS
- Implement appropriate security measures for secure Websockets communication

HTTP/TLS

- TLS/SSL enables encryption over insecure communication channels
- Common application: secure communication for HTTP (HTTPS)
- TLS/SSL ensures privacy, authentication, and data integrity
- Go language has implementation in the crypto/tls package
- Focus on understanding and utilizing Go's TLS/SSL implementation
- Additional information on cryptographic practices in Cryptographic Practices module

HTTP/TLS

```
import "log"
import "net/http"
func main() {
  http.HandleFunc("/", func (w http.ResponseWriter, req *http.Request) {
    w.Write([]byte("This is an example server.\n"))
  })
  // yourCert.pem - path to your server certificate in PEM format
  // yourKey.pem -  path to your server private key in PEM format
  log.Fatal(http.ListenAndServeTLS(":443", "yourCert.pem", "yourKey.pem", nil))
}
```

HTTP/TLS

```
import "log"
import "net/http"
func main() {
  http.HandleFunc("/", func (w http.ResponseWriter, req *http.Request) {
    w.Write([]byte("This is an example server.\n"))
    w.Header().Add("Strict-Transport-Security", "max-age=63072000; includeSubDomains")
  })
  // yourCert.pem - path to your server certificate in PEM format
  // yourKey.pem -  path to your server private key in PEM format
  log.Fatal(http.ListenAndServeTLS(":443", "yourCert.pem", "yourKey.pem", nil))
}
```

Server Name Indication (SNI)

- Go's crypto/tls package implements TLS
- Use consistent and properly configured TLS implementation
- Server Name Indication (SNI) extension allows hostname indication
- SNI enables hosting multiple websites with different certificates
- Client includes requested hostname in ClientHello message
- Server selects certificate based on requested hostname from tls.Config

Server Name Indication (SNI)

```
import "log"
import "net/http"
func main() {
  http.HandleFunc("/", func (w http.ResponseWriter, req *http.Request) {
    w.Write([]byte("This is an example server.\n"))
    w.Header().Add("Strict-Transport-Security", "max-age=63072000; includeSubDomains")
  })
  // yourCert.pem - path to your server certificate in PEM format
  // yourKey.pem -  path to your server private key in PEM format
  log.Fatal(http.ListenAndServeTLS(":443", "yourCert.pem", "yourKey.pem", nil))
}
```

Important Considerations

- Use valid and correct TLS certificates from trusted authorities
- Ensure certificates are not expired; renew when necessary
- Install intermediate certificates for a valid trust chain
- Never set InsecureSkipVerify to true in production environment
- Follow recommended practices for TLS implementation security
- Reject invalid TLS certificates to maintain integrity and security

Production Environment Configuration

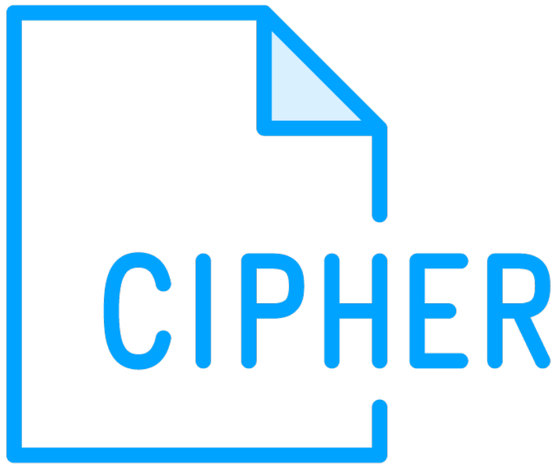
```
config := &tls.Config{InsecureSkipVerify: false}
config := &tls.Config{ServerName: "yourHostname"}
```

POODLE

- POODLE attack: TLS vulnerability with cipher downgrade
- Go proactively disables SSLv3 to prevent POODLE exploitation
- SSLv3 is highly susceptible to the POODLE attack

TLS Cipher Versions

```
// MinVersion contains the minimum SSL/TLS version that is acceptable.
// If zero, then TLS 1.0 is taken as the minimum.
MinVersion uint16
// MaxVersion contains the maximum SSL/TLS version that is acceptable.
// If zero, then the maximum version supported by this package is used,
// which is currently TLS 1.2.
MaxVersion uint16
```



Cipher Security

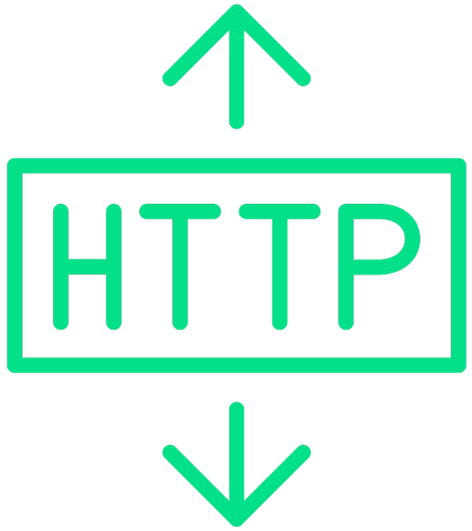
- Use SSL Labs to assess security and status of ciphers
- TLS_FALLBACK_SCSV flag helps mitigate downgrade attacks (not implemented in Go)
- Go client does not engage in fallback behavior by design
- SSL Labs and industry practices ensure secure ciphers adoption
- Assess cipher status with SSL Labs
- TLS_FALLBACK_SCSV flag not needed in Go

CRIME

- CRIME attack targets compressed TLS sessions
- Go programs are typically not vulnerable to CRIME due to lack of compression support
- Beware of compression enabled by external security libraries in Go
- Consider connection renegotiation in TLS
- Use GetClientCertificate function and evaluate associated error code
- Prevent insecure channel usage by capturing and analyzing error codes

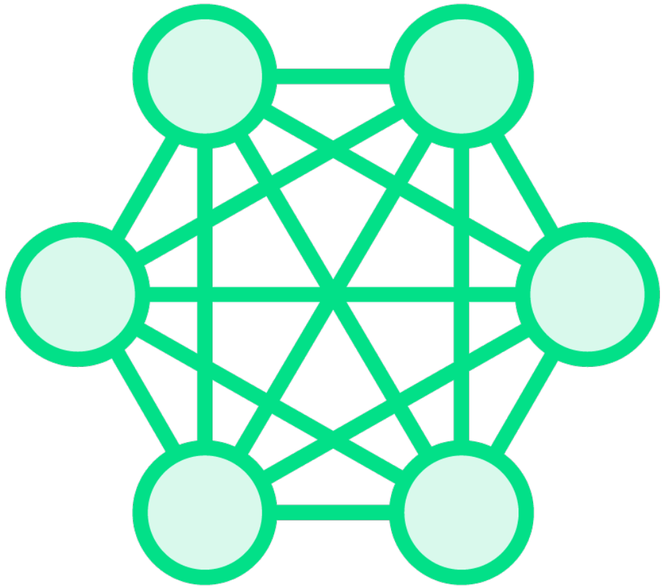
Character Encoding

```
w.Header().Set("Content-Type", "Desired Content Type; charset=utf-8")
```



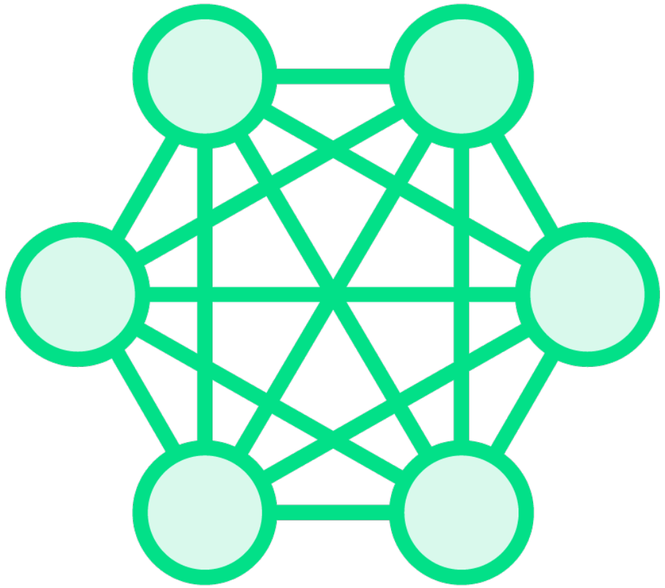
HTTP Headers

- Avoid including sensitive information in HTTP headers
- HTTP headers may inadvertently disclose sensitive data
- Sensitive details like authentication tokens or PII should never be included
- Use encryption mechanisms like HTTPS for secure transmission
- Be cautious about the information contained in HTTP headers
- Prevent exposure of sensitive data through HTTP headers



WebSockets

- WebSocket enables real-time bidirectional communication in HTML5 applications
- Establishes a persistent connection over a single TCP socket
- Eliminates the need for repeated HTTP requests and responses
- Initial HTTP handshake followed by an upgrade to WebSocket connection
- Enables data exchange without additional handshakes
- Enhances interactivity and responsiveness of web applications

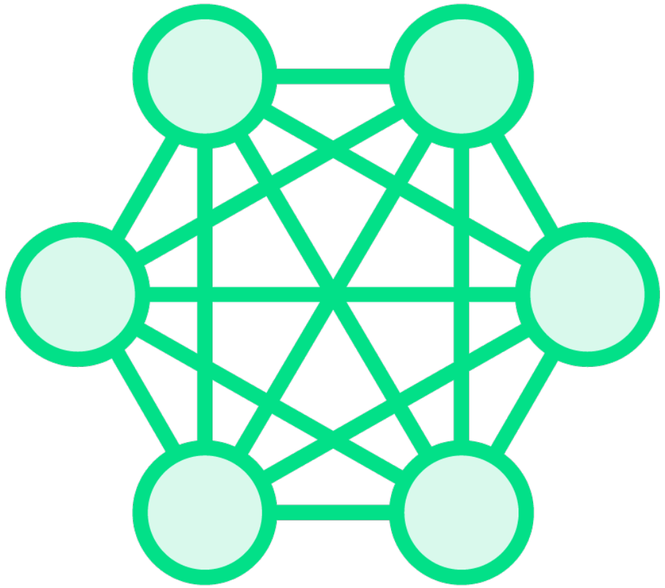


Origin Header

- Origin header verifies the trusted domain in WebSocket connections
- Failure to validate Origin header can lead to CSRF vulnerabilities
- Server must validate the Origin header during the handshake
- Validate both Host and Origin headers to prevent CSWSH attacks
- Reject connections if Origin does not match the trusted Host
- Implementing this validation enhances WebSocket security

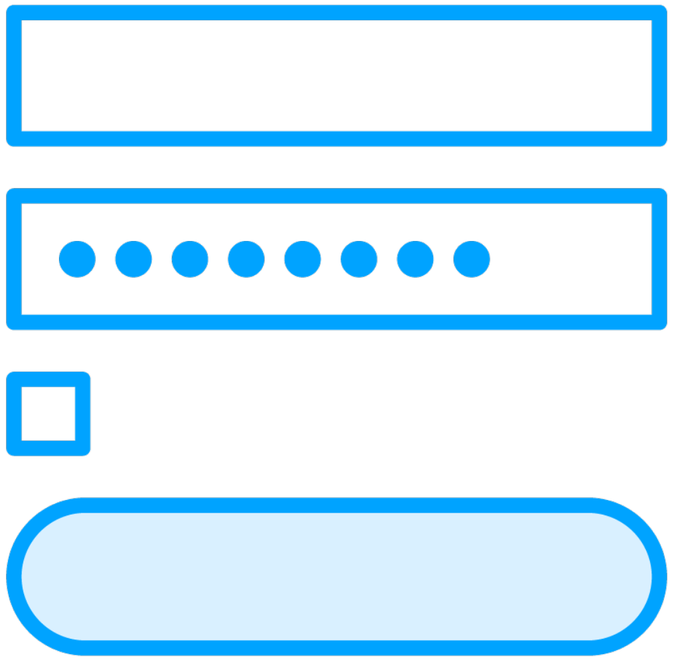
Origin Header Validation

```
//Compare our origin with Host and act accordingly
if r.Header.Get("Origin") != "http://"+r.Host {
http.Error(w, "Origin not allowed", 403)
    return
} else {
    websocket.Handler(EchoHandler).ServeHTTP(w, r)
}
```



Confidentiality and Integrity

- Mishandling header information during the upgrade can pose risks
- Upgrade from HTTP to WebSocket involves a handshake process
- Information exposure: Sensitive data can be exposed to attackers
- MITM attacks: Insecure WebSocket connection due to manipulated headers
- Protocol mismatch: Incompatible communication and potential vulnerabilities
- Mitigation measures: Sanitize and validate headers, use secure connections, handle errors



Authentication and Authorization

- WebSockets require additional mechanisms for authentication and authorization
- Additional mechanisms can include cookies, HTTP authentication, or TLS authentication
- Authentication ensures proper identification and verification of parties involved
- Authorization determines access privileges for authenticated entities
- Comprehensive information available in Authentication and Password Management module
- Access Control module provides techniques and best practices for securing WebSockets

Input Sanitization

- Properly sanitize and validate input data from untrusted sources to remove potentially harmful elements
- Apply output encoding techniques to neutralize special characters or entities in the output and prevent attacks like cross-site scripting (XSS)
- Refer to the Input Validation and Output Encoding modules for detailed information and code samples on implementing these security measures

Summary

Communication Security

- TLS/SSL: Privacy, authentication, and data integrity
- Go empowers robust TLS/SSL implementation
- WebSockets: Real-time bidirectional communication
- Persistent connection, eliminating repeated HTTP requests
- WebSocket protocol boosts interactivity, responsiveness
- Protect information, establish trust, ensure security